

Adding Rigorous Statistics to the Java Benchmarkers' Toolbox

Andy Georges Dries Buytaert Lieven Eeckhout

ELIS, Ghent University, Belgium

{ageorges,dbuytaer,leeckhou}@elis.ugent.be

Abstract

Java performance is far from trivial to benchmark because it is affected by various factors such as the Java application, its input, the virtual machine, the garbage collector, the heap size, etc. In addition, non-determinism due to Just-in-Time compilation/optimization, thread scheduling, etc., causes the execution time of a Java program to differ from run to run.

This poster advocates statistically rigorous data analysis when reporting Java performance. We advise to model non-determinism by computing confidence intervals. In addition, we show that prevalent data analysis approaches may lead to misleading or even incorrect conclusions. Although we focus on Java performance, the techniques can be readily applied to any managed runtime system.

Categories and Subject Descriptors D.2.8 [Software Engineering]: Metrics—Performance measures; D.3.4 [Programming Languages]: Processors—Run-time environments

General Terms Experimentation, Measurement, Performance

Keywords Java, benchmarking, data analysis, methodology, statistics

1. Introduction

Benchmarking is at the heart of experimental computer science research and development. Hence, it is crucial to have a rigorous benchmarking methodology. Otherwise, the overall performance picture may be skewed, and incorrect conclusions may be drawn. In particular, a managed runtime system, such as a Java virtual machine, poses a great challenge to benchmark because there are numerous factors affecting performance, and some of them interact with each other in a non-deterministic way, which is illustrated in a number of research papers [4, 5, 7]. Recent work in Java performance analysis [1, 2] highlights the need for a well chosen and motivated experimental design.

Through an extensive literature survey including 50 papers from the past 7 OOPSLA, PLDI, VEE, ISMM and CGO

conferences, we found that there are many prevalent data analysis approaches. Some report a best value of k benchmark runs, others report a mean value of k runs, yet still others report a second best, or the worst performance number as done by SPEC [8]. In this poster we show that these prevalent data analysis approaches often lead to misleading conclusions, and in some cases even incorrect conclusions.

We advocate the use of statistically rigorous data analysis which computes confidence intervals when reporting experimental results. In addition, we develop a toolbox that automates the measurement and collection of startup and steady-state Java performance numbers.

2. A practical statistically rigorous methodology

In [3], we present the proposed statistically rigorous data analysis methodology in great detail. In this poster, we limit ourselves to presenting the key idea and a few highlights, and refer to the full paper version for an elaborate discussion on the methodology and its underlying statistics.

For Java (startup) performance¹, we propose the following approach to computing a confidence interval: (i) measure the execution time of $n > 1$ VM invocations, running a single benchmark iteration per VM invocation, and (ii) determine a confidence interval for the execution time, using the Student t -statistic² $[\bar{x} \pm t_{\alpha/2;n-1} \frac{s}{\sqrt{n}}]$ [6], where \bar{x} denotes the mean, and s denotes the standard deviation of the collected samples; $t_{\alpha/2;n-1}$ can be found in a precomputed t -statistics table. α is the significance level; $(1 - \alpha)$ is called the confidence level. The meaning of a confidence interval is as follows. A 90% confidence interval, i.e., a confidence interval with a 90% confidence level, means that there is a 90% probability that the actual mean of the underlying population, μ , falls within the confidence interval. It is important to note that computing confidence intervals builds on the assumption that the measurements' mean \bar{x} is Gaussian distributed, which is a valid assumption based on the central limit theory, irrespective of the distribution from which the measurements are taken. In other words, the measurements themselves need not be Gaussian distributed in order to apply statistically rigorous data analysis.

¹We consider quantifying steady-state performance in the full paper version.

²If $n > 30$, one can also use the z -statistic derived from the normal distribution.

Confidence intervals enable quantifying performance in the presence of non-determinism. This is extremely valuable when evaluating design alternatives, i.e., it enables the benchmarker to discern actual performance differences from performance differences due to random fluctuations. When confidence intervals overlap, we conclude that the differences seen in the mean values are possibly due to random effects. If the confidence intervals do not overlap, however, we conclude that there is no evidence to suggest that there is not a statistically significant difference. Or, intuitively speaking, the performance difference is likely a ‘real’ performance difference, not due to random fluctuations. We advocate using an ANOVA and a Tukey’s HSD post hoc test to determine whether more than two design alternatives yield statistically significant performance differences.

3. Evaluation

We evaluate our approach by comparing the conclusions made by prevalent methods to those made by using the statistically rigorous approach outlined in the previous section. The experiments were conducted on an AMD Athlon XP 3000+, running Linux 2.6.18. We use Jikes RVM (SVN version from February 12th, 2007) running the benchmarks from the SPECjvm98 and DaCapo (version 2006-10-MR2) [1] suites. As a case study, we compare the performance of five MMTk garbage collectors: CopyMS, GenCopy, GenMS, MarkSweep and SemiSpace, across a range of heap sizes.

To model the threshold at which people assume prevalent techniques indicate a significant performance difference between alternatives, we introduce the parameter θ . Differences smaller than θ are assumed to be insignificant for a prevalent approach.

Figure 1 shows the percentage GC comparisons by various prevalent data analysis approaches leading to indicative, misleading and incorrect conclusions for the $\theta = 1\%$ threshold. There are bars for reporting the best, the second best, the worst, the mean and the median performance number; and this is done for 3, 5, 10 and 30 VM invocations and a single benchmark iteration — for example, the ‘best of 3’ means taking the best performance number out of 3 VM invocations. The statistically rigorous methodology we compare against, considers 30 VM invocations and a single benchmark iteration per VM invocation, and considers 95% confidence intervals, using an ANOVA³ and a Tukey’s HSD post hoc test on a per-heap-size basis.

We can draw several important conclusions from this analysis. First of all, for a substantial percentage of comparisons, prevalent methods are misleading: for over 10% of the comparisons, the prevalent approach concludes there is a significant performance difference although there is no performance difference when using rigorous statistics. Second, for a fair number of comparisons, a prevalent methodology can lead to incorrect conclusions: in up to 3% of the comparisons, the prevalent approach concludes one alternative

³This test is robust against departure from normality.

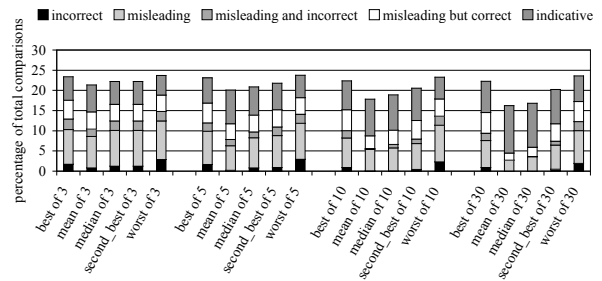


Figure 1. Percentage GC comparisons by prevalent data analysis approaches leading to incorrect, misleading or indicative conclusions. Results are shown for the AMD Athlon machine with $\theta = 1\%$

is better than the another, however, rigorous statistics concludes the opposite.

4. Summary

Prevalent data analysis approaches deal with the non-determinism in Java systems in a wide variety of ways. This poster shows that prevalent data analysis approaches can be misleading and can even lead to incorrect conclusions. This poster advocated statistically rigorous data analysis for quantifying Java performance. The key point is to deal with the non-determinism by computing confidence intervals. For a more elaborate description, we refer to [3].

References

- [1] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. Eliot, B. Moss, A. Phansalkar, D. Stefanovic, T. VanDrunen, D. von Dincklage, and B. Wiedermann. The DaCapo benchmarks: Java benchmarking development and analysis. In *OOPSLA*, pages 169–190, Oct. 2006.
- [2] L. Eeckhout, A. Georges, and K. De Bosschere. How Java programs interact with virtual machines at the microarchitectural level. In *OOPSLA*, pages 169–186, Nov. 2003.
- [3] A. Georges, D. Buytaert, L. Eeckhout. Statistically Rigorous Java Performance Evaluation. In *OOPSLA*, Oct. 2007.
- [4] D. Gu, C. Verbrugge, and E. M. Gagnon. Relative factors in performance analysis of Java virtual machines. In *VEE*, pages 111–121, June 2006.
- [5] M. Hauswirth, P. F. Sweeney, A. Diwan, and M. Hind. Vertical profiling: Understanding the behavior of object-oriented applications. In *OOPSLA*, pages 251–269, Oct. 2004.
- [6] D. J. Lilja. *Measuring Computer Performance: A Practitioner’s Guide*. Cambridge University Press, 2000.
- [7] J. Maebe, D. Buytaert, L. Eeckhout, and K. De Bosschere. Javana: A system for building customized Java program analysis tools. In *OOPSLA*, pages 153–168, Oct. 2006.
- [8] Standard Performance Evaluation Corporation. SPECjvm98 Benchmarks. <http://www.spec.org/jvm98/>.